09/667430

Cof C

ATTORNEY DOCKET 728-170
PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicants: Koved et al.

Patent No: 6,925,638

Issued: August 2, 2005

For: MUTABILITY ANALYSIS IN JAVA

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

REQUEST FOR CERTIFICATE OF CORRECTION
OF PATENT FOR PTO MISTAKE UNDER 37 C.F.R. §1.322

Sir:

The patentees hereby request the issuance of a Certificate of Correction to correct errors originating at the U.S. Patent Office in Claims 1, 7, 8, 19, 26, 38, 44, 45, 53 and 60.

The claims are listed with the correct wording on attached Form PTO/SB/44.

It appears that the amendments to the claims in an Amendment filed October 15, 2004 while entered by the Examiner and allowed as indicated in the Notice of Allowability dated February 16, 2005, were not included in the printed Patent. A copy of the Amendment is enclosed your convenience.

This form is being sent in duplicate with at least one copy being suitable for printing.

CERTIFICATE OF MAILING UNDER 37 C.F.R. § 1.8(a)

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on September 23, 2005.

Dated: September 23, 2005

Paul J. Farrell

It is Applicants' belief that these errors originated at the U.S. Patent and Trademark Office, and accordingly, no fee is believed to be required. Please charge any fees that may become due to Deposit Account No. **50-0510 (IBM)**. A duplicate copy of this sheet is enclosed.

Respectfully submitted,

Paul J. Farrell
Registration No. 33,494
Attorney for Applicant/s

DILWORTH & BARRESE, LLP
333 Earle Ovington Boulevard
Uniondale, New York 11553
(516) 228-8484

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : **6,925,638**

DATED : **August 2, 2005**

INVENTOR(S) : **Koved et al.**

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 1 should read as follows:

1. A method of detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the method comprising the steps of:

determining whether any variable in the program component could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

performing encapsulation analysis to determine whether any variable in the program component could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component, to identify an exposure of the variables of the program component to modification external to the program component,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

an object is mutable if its state ever changes after said object is initialized, said state of said object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable.

6,925,638

PATENT NO. _____

MAILING ADDRESS OF SENDER:

Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 7 should read as follows:

7. A method of detecting mutability of classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the method comprising the steps of:
obtaining a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;
testing each undecided class, said test being comprised of the sub-steps of:
testing each field in said undecided class being tested, said test being comprised of the sub-sub-steps of:
determining whether any variable corresponding to said each field could undergo a state modification of first type, said first type state modification being made by at least one method that is within said component; and
performing encapsulation analysis to determine whether any variable corresponding to said each field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within said component;
classifying said each field as immutable if no possible first type or second type state modifications are found;
classifying said each field as undecided if there is insufficient class mutability information; and
classifying said each field as mutable otherwise;
re-classifying said undecided class as mutable if any fields in said undecided class are mutable;
re-classifying said undecided class as immutable if all fields in said undecided class are mmutable;
repeating said testing each undecided class step until a number of undecided classes after a repetition of said testing step is identical to a number of undecided classes before the repetition of said testing step; and
re-classifying remaining undecided classes as mutable classes, to identify an exposure of variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:
Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 8 should read as follows:

8. A method of detecting mutability of classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the method comprising the steps of:

obtaining a set of classes, each of said classes being classified as one of mutable, immutable, and undecided; testing each undecided class, said test being comprised of the sub-steps of:

testing each instance field in said undecided class being tested, said test being comprised of the sub-sub-steps of:

determining whether any variable corresponding to said each instance field could undergo a state modification of first type, said first type state modification being made by at least one method that is within said component; and

performing encapsulation analysis to determine whether any variable corresponding to said each instance field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within said component;

classifying said each instance field as immutable if no possible first type or second type state modifications are found;

classifying said each instance field as undecided if there is insufficient class mutability information; and

classifying said each instance field as mutable otherwise;

re-classifying said undecided class as mutable if any instance fields in said undecided class are mutable;

re-classifying said undecided class as immutable if all instance fields in said undecided class are immutable;

repeating said testing each undecided class step until a number of undecided classes after a repetition of said testing step is identical to a number of undecided classes before the repetition of said testing step; and

re-classifying remaining undecided classes as mutable classes to identify an exposure of variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:

Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 19 should read as follows:

19. A method of detecting mutability of classes and class variables in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming, the method comprising the steps of:
obtaining a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;
testing each undecided class, said test being comprised of the sub-steps of:
testing mutability of each instance field in said undecided class being tested;
classifying an instance field as immutable if no possible state or encapsulation analysis modifications are found;
classifying an instance field as undecided if there is insufficient class mutability information; and
classifying an instance field as mutable otherwise;
re-classifying an undecided class as mutable if any instance fields in said undecided class are mutable;
re-classifying said undecided class as immutable if all instance fields in said undecided class are immutable;
repeating said testing each undecided class step until a number of undecided classes after a repetition of said testing step is identical to a number of undecided classes before the repetition of said testing step;
re-classifying remaining undecided classes as mutable classes; and
testing mutability of each class field in each class to identify an exposure of variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:
Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 26 should read as follows:

26. A device for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the device comprising: memory for holding

a layer of at least one core library and at least one data-flow analysis engine, for providing a particular abstraction of the program component;

a layer of at least one utility module, for using the results of the at least one data analysis engine to generate basic results; and

a layer of at least one mutability sub-analysis module for generating final results,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of a variable being its value together with a state of any referenced object,

an object is mutable if its state ever changes after said object is initialized, the state of an object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable to identify an exposure of the variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:

Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 38 should read as follows:

38. A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the computer system comprising:

at least one computer-readable memory including:

code that determines whether any variable in the program component could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

code that performs encapsulation analysis to determine whether any variable in the program component could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component to identify an exposure of the variables of the program component to modification external to the program component,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

an object is mutable if its state ever changes after said object is initialized, the state of said object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable.

MAILING ADDRESS OF SENDER:

Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 44 should read as follows:

44. A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the computer system comprising:

at least one computer-readable memory including:

code that obtains a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;

code that tests each undecided class, said test being comprised of:

code that tests each field in said undecided class being tested, said field testing code being comprised of:

code that determines whether any variable corresponding to said each field could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

code that performs encapsulation analysis to determine whether any variable corresponding to said each field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component;

code that classifies said each field as immutable if no possible state modifications or breakages of encapsulation are found;

code that classifies said each field as mutable if possible state modifications or breakages of encapsulation are found; and

code that classifies said each field as undecided if there is insufficient class mutability information;

code that re-classifies said undecided class as mutable if any fields in said undecided class are mutable;

code that re-classifies said undecided class as immutable if all fields in said undecided class are immutable;

code that repeats said testing each undecided class code until a number of undecided classes after a repetition of said testing code is identical to a number of undecided classes before the repetition of said testing code; and

code that re-classifies remaining undecided classes as mutable classes to identify an exposure of the variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:

Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 45 should read as follows:

45. A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being written in an object-oriented programming language, the computer system comprising:
at least one computer-readable memory including:
code that obtains a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;
code that tests each undecided class, said test being comprised of:
code that tests each instance field in said undecided class being tested, said instance field testing code being comprised of:
code that determines whether any variable corresponding to said each instance field could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and
code that performs encapsulation analysis to determine whether any variable corresponding to said each instance field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component;
code that classifies said each instance field as immutable if no possible state modifications or breakages of encapsulation are found;
code that classifies said each instance field as mutable if possible state modifications or breakages of encapsulation are found; and
code that classifies said each instance field as undecided if there is insufficient class mutability information;
code that re-classifies said undecided class as mutable if any instance fields in said undecided class are mutable;
code that re-classifies said undecided class as immutable if all instance fields in said undecided class are immutable;
code that repeats said testing each undecided class code until a number of undecided classes after a repetition of said testing code is identical to a number of undecided classes before the repetition of said testing code; and
code that re-classifies remaining undecided classes as mutable classes to identify an exposure of the variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:
Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 53 should read as follows:

53. A computer system for detecting mutability of classes and class variables in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the computer system comprising:
at least one computer-readable memory including:
code that obtains a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;
code that tests each undecided class, said test being comprised of:
code that tests each instance field in said undecided class being tested, said instance field testing code being comprised of:
code that determines whether any variable corresponding to said each instance field could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and
code that performs encapsulation analysis to determine whether any variable corresponding to said each instance field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component;
code that classifies said each instance field as immutable if no possible state modifications or breakages of encapsulation are found;
code that classifies said each instance field as mutable if possible state modifications or breakages of encapsulation are found; and
code that classifies said each instance field as undecided if there is insufficient class mutability information;
code that re-classifies said undecided class as mutable if any instance fields in said undecided class are mutable;
code that re-classifies said undecided class as immutable if all instance fields in said undecided class are immutable;
code that repeats said testing each undecided class code until a number of undecided classes after a repetition of said testing code is identical to a number of undecided classes before the repetition of said testing code;
code that re-classifies remaining undecided classes as mutable classes; and
code that tests mutability of each class field in each class to identify an exposure of the variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:
Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

PTO/SB/44 (02-01)
Approved for use through 01/31/2004. OMB 0651-0033
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

(Also Form PTO-1050)

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO : 6,925,638

DATED : August 2, 2005

INVENTOR(S) : Koved et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Claim 60 should read as follows:

60. A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being created in an object-oriented programming language, the computer system comprising:
at least one computer-readable memory including:
code that maintains a layer of at least one core library and at least one data-flow analysis engine in a mutability analyzer, for providing a particular abstraction of the program component;
code that maintains a layer of at least one utility module in a mutability analyzer, for using the results of the at least one data analysis engine to generate basic results; and
code that maintains a layer of at least one mutability sub-analysis module in a mutability analyzer, for generating final results,
wherein a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,
an object is mutable if its state ever changes after said object is initialized, the state of said object being a set of states of all associated variables,
a field is mutable if any variable corresponding to said field is mutable, and
a class is mutable if any instance fields implemented by said class are mutable to identify an exposure of the variables of the program component to modification external to the program component.

MAILING ADDRESS OF SENDER:
Paul J. Farrell, Esq.
Dilworth & Barrese LLP
333 Earle Ovington Blvd.
Uniondale, New York 11553

PATENT NO. 6,925,638

No. of additional copies

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICANTS:	Larry Koved, et al.	**EXAMINER:**	Trenton J. Roche
SERIAL NO.:	09/667,430	**GROUP ART UNIT**	2124
FILED:	September 21, 2000	**DATED:**	October 15, 2004
FOR:	**MUTABILITY ANALYSIS IN JAVA**		

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

AMENDMENT

Sir:

In response to the Office Action of the United States Patent and Trademark Office mailed July 15, 2004, please consider the following amendments and remarks.

CERTIFICATE OF MAILING UNDER 37 C.F.R. §1.10

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail, postpaid in an envelope, addressed to the: Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date set forth below.

Dated: October 15, 2004

Michael J. Musella

OCT 3 2005

IN THE CLAIMS

1. (Currently Amended) A method of detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the method comprising the steps of:

determining whether any variable in the program component could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

performing encapsulation analysis to determine whether any variable in the program component could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component, to identify an exposure of the variables of the program component to modification external to the program component,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

~~wherein~~ an object is mutable if its state ever changes after said object is initialized, said state of said object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable.

2. (Original) The method as recited in claim 1, the first type state modification determination step comprising the steps of:

detecting possible first type state modification of a value held in said each variable; and

detecting possible first type state modification of a state of any object referenced by said each variable.

3. (Previously Presented) The method as recited in claim 1, the encapsulation analysis step comprising the steps of:

detecting possible second type state modification of a value held in said each variable;

detecting possible second type state modification of a state of any object referenced by

said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

detecting possible breakage of variable encapsulation,

wherein a variable is encapsulated if all references to objects reachable from it are defined within the program component and variable encapsulation is broken if a method within the program component causes a mutable object reachable from the variable to become accessible to at least one method that is not within the program component.

4. (Previously Presented) The method as recited in claim 1, wherein the method is implemented in an object oriented environment, said any instance fields being non-static fields, said variables being class variables or instance variables, each of said class variables being initialized upon completion of its corresponding <clinit> method, and each of said instance variables being initialized upon completion of its corresponding <init> method.

5. (Original) The method as recited in claim 1, further comprising the step of: identifying isolation faults due to detected mutable global variables or objects.

6. (Original) The method as recited in claim 1, further comprising the step of: identifying fields and objects that can be determined to be constants because said identified fields and objects are not in the set of detected mutable fields and objects.

7. (Currently Amended) A method of detecting mutability of classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the method comprising the steps of:

obtaining a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;

testing each undecided class, said test being comprised of the sub-steps of:

testing each field in said undecided class being tested, said test being comprised of the sub-sub-steps of:

determining whether any variable corresponding to said each field could undergo a state modification of first type, said first type state modification being made by at least one method that is within said component; and

performing encapsulation analysis to determine whether any variable corresponding to said each field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within said component;

classifying said each field as immutable if no possible first type or second type state modifications are found;

classifying said each field as undecided if there is insufficient class mutability information; and

classifying said each field as mutable otherwise;

re-classifying said undecided class as mutable if any fields in said undecided class are mutable;

re-classifying said undecided class as immutable if all fields in said undecided class are immutable;

repeating said testing each undecided class step until a number of undecided classes after a repetition of said testing step is identical to a number of undecided classes before the repetition of said testing step; and

re-classifying remaining undecided classes as mutable classes, to identify an exposure of variables of the program component to modification external to the program component.

8. (Currently Amended) A method of detecting mutability of classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the method comprising the steps of:

obtaining a set of classes, each of said classes being classified as one of mutable, immutable, and undecided; testing each undecided class, said test being comprised of the sub-steps of:

testing each instance field in said undecided class being tested, said test being

comprised of the sub-sub-steps of:

determining whether any variable corresponding to said each instance field could undergo a state modification of first type, said first type state modification being made by at least one method that is within said component; and

performing encapsulation analysis to determine whether any variable corresponding to said each instance field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within said component;

classifying said each instance field as immutable if no possible first type or second type state modifications are found;

classifying said each instance field as undecided if there is insufficient class mutability information; and

classifying said each instance field as mutable otherwise;

re-classifying said undecided class as mutable if any instance fields in said undecided class are mutable;

re-classifying said undecided class as immutable if all instance fields in said undecided class are immutable;

repeating said testing each undecided class step until a number of undecided classes after a repetition of said testing step is identical to a number of undecided classes before the repetition of said testing step; and

re-classifying remaining undecided classes as mutable classes to identify an exposure of variables of the program component to modification external to the program component.

9. (Previously Presented) The method as recited in claim 8, the first type state modification determination sub-sub-step comprising the steps of:

detecting possible first type state modification of a value held in said each variable; and

detecting possible first type state modification of a state of any object referenced by said each variable,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables and

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object.

10. (Previously Presented) The method as recited in claim 8, the performing encapsulation analysis sub-sub-step comprising the steps of:

detecting possible second type modification of a value of said each variable;

detecting possible second type modification of a state of any object referenced by said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

detecting possible breakage of variable encapsulation,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables,

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

a variable is encapsulated if all references to objects reachable from it are defined within said component, and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from a variable to become accessible to at least one method that is not within said component.

11. (Previously Presented) The method as recited in claim 8, wherein the method is implemented in an object oriented environment, said each variable corresponding to said each instance field being a non-static variable, and each non-static variable being initialized upon completion of its corresponding <init> method.

12. (Original) The method as recited in claim 8, further comprising the steps of:

identifying an object as mutable if it is an instance of a mutable class;

identifying an object as immutable if it is an instance of an immutable class; and

identifying fields and objects that can be determined to be constants because said identified fields and objects are not in a set of detected mutable fields and objects.

13. (Original) The method as recited in claim 8, further comprising the step of: testing mutability of each undecided class field in each class.

14. (Original) The method as recited in claim 13, further comprising the step of: identifying isolation faults due to detected mutable class fields.

15. (Original) The method as recited in claim 13, the step of testing mutability of each undecided class field in each class comprising the sub-steps of:

determining whether any variable corresponding to said each undecided class field could undergo a first type state modification; and

performing encapsulation analysis to determine whether any variable corresponding to said each undecided class field could undergo a second type state modification.

16. (Previously Presented) The method as recited in claim 15, wherein the determining whether any variable corresponding to said each undecided class field could undergo a first type state modification sub-step comprises the steps of:

detecting possible first type state modification of a value held in said each variable; and

detecting possible first type state modification of a state of any object referenced by said each variable,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables and

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object.

17. (Previously Presented) The method as recited in claim 15, wherein the performing encapsulation analysis to determine whether any variable corresponding to said each undecided class field could undergo a second type state modification sub-step comprises the steps of:

detecting possible second type state modification of a value of said each variable;

detecting possible second type state modification of a state of any object referenced by

said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

detecting possible breakage of variable encapsulation,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables,

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

a variable is encapsulated if all references to objects reachable from it are defined within said component, and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from a variable to become accessible to at least one method that is not within said component.

18. (Previously Presented) The method as recited in claim 13, wherein the method is implemented in an object oriented environment, said variables corresponding to said each undecided class field being static variables, and each static variables being initialized upon completion of its corresponding <clinit> method.

19. (Currently Amended) A method of detecting mutability of classes and class variables in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming, the method comprising the steps of:

obtaining a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;

testing each undecided class, said test being comprised of the sub-steps of:

testing mutability of each instance field in said undecided class being tested;

classifying an instance field as immutable if no possible state or encapsulation analysis modifications are found;

classifying an instance field as undecided if there is insufficient class mutability information; and

classifying an instance field as mutable otherwise;

re-classifying an undecided class as mutable if any instance fields in said undecided class are mutable;

re-classifying said undecided class as immutable if all instance fields in said undecided class are immutable;

repeating said testing each undecided class step until a number of undecided classes after a repetition of said testing step is identical to a number of undecided classes before the repetition of said testing step;

re-classifying remaining undecided classes as mutable classes; and

testing mutability of each class field in each class to identify an exposure of variables of the program component to modification external to the program component.

20. (Original) The method as recited in claim 19, wherein testing mutability of a field, whether said field is an instance field or a class field, is comprised of the sub-steps of:

determining whether any variable corresponding to said field being tested could undergo a state modification of a first type, said first type state modification being made by at least one method that is within said program component; and

performing encapsulation analysis to determine whether any variable corresponding to said field being tested could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within said program component;

classifying said field being tested as immutable if no possible state modifications or breakages of encapsulation are found;

classifying said field being tested as undecided if there is insufficient class mutability information; and

classifying said field being tested as mutable otherwise.

21. (Previously Presented) The method as recited in claim 20, wherein the first type state modification determination sub-step comprises the steps of:

detecting possible first type state modification of a value held in said each variable; and

detecting possible first type state modification of a state of any object referenced by said

each variable,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables and

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object.

22. (Previously Presented) The method as recited in claim 20, wherein the performing encapsulation analysis sub-step comprises the steps of:

detecting possible second type state modification of a value of said each variable;

detecting possible second type state modification of a state of any object referenced by said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

detecting possible breakage of variable encapsulation,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables,

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

a variable is encapsulated if all references to objects reachable from it are defined within said component, and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from a variable to become accessible to at least one method that is not within said component.

23. (Previously Presented) The method as recited in claim 19, wherein the method is implemented in an object oriented environment, said instance fields being non-static fields, an instance variable being initialized upon completion of its corresponding <init> method, and said class fields being static fields, a class variable being initialized upon completion of its corresponding <clinit> method.

24. (Original) The method as recited in claim 19, further comprising the steps of:

identifying an object as mutable if it is an instance of a mutable class;

identifying an object as immutable if it is an instance of an immutable class; and

identifying fields and objects that can be determined to be constants because said identified fields and objects are not in a set of detected mutable fields and objects.

25. (Original) The method as recited in claim 19, further comprising the step of:

identifying isolation faults due to detected mutable class fields.

26. (Currently Amended) A device for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the device comprising: memory for holding

a layer of at least one core library and at least one data-flow analysis engine, for providing a particular abstraction of the program component;

a layer of at least one utility module, for using the results of the at least one data analysis engine to generate basic results; and

a layer of at least one mutability sub-analysis module for generating final results,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of a variable being its value together with a state of any referenced object,

an object is mutable if its state ever changes after said object is initialized, the state of an object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable to identify an exposure of the variables of the program component to modification external to the program component.

27. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one core library and at least one data analysis engine comprising:

a library for collecting and manipulating static information about the program component by analyzing a set of class files, and for effectively constructing the program component's

OCT 3 2005

reference, hierarchy, and call graphs.

28. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one core library and at least one data analysis engine comprising: a library for allowing a user to read class_files.

29. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one core library and at least one data analysis engine comprising: an intra-procedural data analysis engine for iteratively computing an effect of an instruction on information associated with locations on a method frame, said method frame being an operand stack and a local variables array.

30. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one core library and at least one data analysis engine comprising: an inter-procedural data analysis engine for computing the effect of a method on information associated with variables that still exist upon completion of this method.

31. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one utility module comprising: a type analysis utility module for identifying a set of possible types for each instruction and each frame location in each method.

32. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one utility module comprising: a reachability analysis utility module for identifying, for each method, a set of escaping objects and class variables from which a mutable object is reachable for each instruction and each frame location referring to said mutable object.

33. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one mutability sub-analysis module comprising: a value modification mutability sub-analysis module for identifying, for each method, a set of fields whose corresponding instance and class variables may be set within said each method.

34. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one mutability sub-analysis module comprising: an object modification mutability sub-analysis module for identifying, for each method, a set of reference-type fields and method parameters, said set of reference-type fields and method parameters referencing an object, a state of said object being modified by said each method.

35. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one mutability sub-analysis module comprising: a variable accessibility mutability sub-analysis module for identifying, for each variable, whether its value may be modified directly by at least one method that is not within the program component.

36. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one mutability sub-analysis module comprising: an object accessibility mutability sub-analysis module for detecting possible accessibility of a state of each object, by determining if each variable associated with said object is encapsulated,

wherein a variable is encapsulated if all references to objects reachable from it are defined within the program component;, and

said accessibility is made by at least one method that is not within the program component.

37. (Previously Presented) The device as recited in claim 26, wherein the layer of at least one mutability sub-analysis module comprising: a breakage of encapsulation mutability sub-analysis module for detecting a possible breakage of encapsulation,

wherein a variable is encapsulated if all references to mutable objects reachable from it are defined within the program component, and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from the variable to become accessible to at least one method that is not within the program component.

38. (Currently Amended) A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the computer system comprising:

at least one computer-readable memory including:

code that determines whether any variable in the program component could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

code that performs encapsulation analysis to determine whether any variable in the program component could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component to identify an exposure of the variables of the program component to modification external to the program component,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

an object is mutable if its state ever changes after said object is initialized, the state of said object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable.

39. (Original) The computer system as recited in claim 38, wherein the code that determines whether any variable could undergo the first type state modification comprises:

code that detects possible first type state modification of a value held in said each variable; and

code that detects possible first type state modification of a state of any object referenced by said each variable.

40. (Previously Presented) The computer system as recited in claim 38, wherein the code that performs encapsulation analysis step comprises:

code that detects possible second type state modification of a value held in said each variable;

code that detects possible second type state modification of a state of any object referenced by said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

code that detects possible breakage of variable encapsulation,

wherein a variable is encapsulated if all references to objects reachable from it are defined within the program component and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from the variable to become accessible to at least one method that is not within the program component.

41. (Previously Presented) The computer system as recited in claim 38, wherein the program component is implemented in an object oriented environment, said any instance fields being non-static fields, said variables being class variables or instance variables, each of said class variables being initialized upon completion of its corresponding <clinit> method, and each of said instance variables being initialized upon completion of its corresponding <init> method.

42. (Previously Presented) The computer system as recited in claim 38, wherein the at least one computer-readable memory further includes: code that identifies isolation faults due to detected mutable global variables or objects.

43. (Original) The computer system as recited in claim 38, wherein the at least one computer-readable memory further includes: code that identifies fields and objects that can be determined to be constants because said identified fields and objects are not in the set of detected mutable fields and objects.

44. (Currently Amended) A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming

language, the computer system comprising:

at least one computer-readable memory including:

code that obtains a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;

code that tests each undecided class, said test being comprised of:

code that tests each field in said undecided class being tested, said field testing code being comprised of:

code that determines whether any variable corresponding to said each field could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

code that performs encapsulation analysis to determine whether any variable corresponding to said each field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component;

code that classifies said each field as immutable if no possible state modifications or breakages of encapsulation are found;

code that classifies said each field as mutable if possible state modifications or breakages of encapsulation are found; and

code that classifies said each field as undecided if there is insufficient class mutability information;

code that re-classifies said undecided class as mutable if any fields in said undecided class are mutable;

code that re-classifies said undecided class as immutable if all fields in said undecided class are immutable;

code that repeats said testing each undecided class code until a number of undecided classes after a repetition of said testing code is identical to a number of undecided classes before the repetition of said testing code; and

code that re-classifies remaining undecided classes as mutable classes to identify an exposure of the variables of the program component to modification external to the program component.

45. (Currently Amended) A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being written ~~created~~ in an object-oriented programming language, the computer system comprising:

at least one computer-readable memory including:

code that obtains a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;

code that tests each undecided class, said test being comprised of:

code that tests each instance field in said undecided class being tested, said instance field testing code being comprised of:

code that determines whether any variable corresponding to said each instance field could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

code that performs encapsulation analysis to determine whether any variable corresponding to said each instance field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component;

code that classifies said each instance field as immutable if no possible state modifications or breakages of encapsulation are found;

code that classifies said each instance field as mutable if possible state modifications or breakages of encapsulation are found; and

code that classifies said each instance field as undecided if there is insufficient class mutability information;

code that re-classifies said undecided class as mutable if any instance fields in said undecided class are mutable;

code that re-classifies said undecided class as immutable if all instance fields in said undecided class are immutable;

code that repeats said testing each undecided class code until a number of undecided

classes after a repetition of said testing code is identical to a number of undecided classes before the repetition of said testing code; and

code that re-classifies remaining undecided classes as mutable classes to identify an exposure of the variables of the program component to modification external to the program component.

46. (Previously Presented) The computer system as recited in claim 45, wherein the code that determines whether any variable could undergo a first type state modification comprises:

code that detects possible first type state modification of a value held in said each variable; and

code that detects possible first type state modification of a state of any object referenced by said each variable,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables and

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object.

47. (Previously Presented) The computer system as recited in claim 45, wherein the code that performs encapsulation analysis comprises:

code that detects possible second type modification of a value of said each variable;

code that detects possible second type modification of a state of any object referenced by said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

code that detects possible breakage of variable encapsulation,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables,

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

a variable is encapsulated if all references to objects reachable from it are defined within

said component, and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from a variable to become accessible to at least one method that is not within said component.

48. (Previously Presented) The computer system as recited in claim 45, wherein the program component is implemented in an object oriented environment, said each variable corresponding to said each instance field being a non-static variable, and each non-static variable being initialized upon completion of its corresponding <init> method.

49. (Original) The computer system as recited in claim 45, wherein the at least one computer-readable memory further includes:

code that identifies an object as mutable if it is an instance of a mutable class;

code that identifies an object as immutable if it is an instance of an immutable class; and

code that identifies fields and objects that can be determined to be constants because said identified fields and objects are not in a set of detected mutable fields and objects.

50. (Original) The computer system as recited in claim 45, wherein the at least one computer-readable memory further includes: code that tests mutability of each undecided class field in each class.

51. (Original) The computer system as recited in claim 50, wherein the at least one computer-readable memory further includes: code that identifies isolation faults due to detected mutable class fields.

52. (Original) The computer system as recited in claim 50, wherein the code that tests mutability of each undecided class field in each class comprises:

code that determines whether any variable corresponding to said each undecided class field could undergo a first type state modification; and

code that performs encapsulation analysis to determine whether any variable

corresponding to said each undecided class field could undergo a second type state modification.

53. (Currently Amended) A computer system for detecting mutability of classes and class variables in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the computer system comprising:

at least one computer-readable memory including:

code that obtains a set of classes, each of said classes being classified as one of mutable, immutable, and undecided;

code that tests each undecided class, said test being comprised of:

code that tests each instance field in said undecided class being tested, said instance field testing code being comprised of:

code that determines whether any variable corresponding to said each instance field could undergo a state modification of a first type, said first type state modification being made by at least one method that is within the program component; and

code that performs encapsulation analysis to determine whether any variable corresponding to said each instance field could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within the program component;

code that classifies said each instance field as immutable if no possible state modifications or breakages of encapsulation are found;

code that classifies said each instance field as mutable if possible state modifications or breakages of encapsulation are found; and

code that classifies said each instance field as undecided if there is insufficient class mutability information;

code that re-classifies said undecided class as mutable if any instance fields in said undecided class are mutable;

code that re-classifies said undecided class as immutable if all instance fields in said undecided class are immutable;

code that repeats said testing each undecided class code until a number of undecided classes after a repetition of said testing code is identical to a number of undecided classes before the repetition of said testing code;

code that re-classifies remaining undecided classes as mutable classes; and

code that tests mutability of each class field in each class to identify an exposure of the variables of the program component to modification external to the program component.

54. (Original) The computer system as recited in claim 53, wherein the code that tests mutability of a field, whether said field is an instance field or a class field, is comprised of:

code that determines whether any variable corresponding to said field being tested could undergo a state modification of a first type, said first type state modification being made by at least one method that is within said program component; and

code that performs encapsulation analysis to determine whether any variable corresponding to said field being tested could undergo a state modification of a second type, said second type state modification being made by at least one method that is not within said program component;

code that classifies said field being tested as immutable if no possible state modifications or breakages of encapsulation are found;

code that classifies said field being tested as undecided if there is insufficient class mutability information; and

code that classifies said field being tested as mutable otherwise.

55. (Previously Presented) The computer system as recited in claim 54, wherein the code that determines whether any variable could undergo first type state modification comprises:

code that detects possible first type state modification of a value held in said each variable; and

code that detects possible first type state modification of a state of any object referenced by said each variable,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables and

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object.

56. (Previously Presented) The computer system as recited in claim 54, wherein the code that performs encapsulation analysis comprises:

code that detects possible second type state modification of a value of said each variable;

code that detects possible second type state modification of a state of any object referenced by said each variable, said possible second type state modification of a state of any object occurring at a point of initialization; and

code that detects possible breakage of variable encapsulation,

wherein a state of an object is modified if it can change after said object is initialized, and the state of an object is a set of states of all associated variables,

a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

a variable is encapsulated if all references to objects reachable from it are defined within said component, and

variable encapsulation is broken if a method within the program component causes a mutable object reachable from a variable to become accessible to at least one method that is not within said component.

57. (Previously Presented) The computer system as recited in claim 53, wherein the program component is implemented in an object oriented environment, said instance fields being non-static fields, an instance variable being initialized upon completion of its corresponding <init> method, and said class fields being static fields, a class variable being initialized upon completion of its corresponding <clinit> method.

58. (Original) The computer system as recited in claim 53, wherein the at least one computer-readable memory further includes:

code that identifies an object as mutable if it is an instance of a mutable class;

code that identifies an object as immutable if it is an instance of an immutable class; and

code that identifies fields and objects that can be determined to be constants because said identified fields and objects are not in a set of detected mutable fields and objects.

59. (Original) The computer system as recited in claim 53, wherein the at least one computer-readable memory further includes: code that identifies isolation faults due to detected mutable class fields.

60. (Currently Amended) A computer system for detecting mutability of variables, objects, fields, and classes in a program component executing on a computing device including a processor and memory, said component being ~~written~~ created in an object-oriented programming language, the computer system comprising:

at least one computer-readable memory including:

code that maintains a layer of at least one core library and at least one data-flow analysis engine in a mutability analyzer, for providing a particular abstraction of the program component;

code that maintains a layer of at least one utility module in a mutability analyzer, for using the results of the at least one data analysis engine to generate basic results; and

code that maintains a layer of at least one mutability sub-analysis module in a mutability analyzer, for generating final results,

wherein a variable is mutable if its state ever changes after said variable is initialized, the state of said variable being its value together with a state of any referenced object,

an object is mutable if its state ever changes after said object is initialized, the state of said object being a set of states of all associated variables,

a field is mutable if any variable corresponding to said field is mutable, and

a class is mutable if any instance fields implemented by said class are mutable to identify an exposure of the variables of the program component to modification external to the program component.

61. (Original) The computer system as recited in claim 60, wherein the code that

OCT 3 2005

OCT 3 2005

maintains the layer of at least one core library and at least one data analysis engine comprises:

code that collects and manipulates static information about the program component by analyzing a set of classfiles; and

code that effectively constructs the program component's reference, hierarchy, and call graphs.

62. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one core library and at least one data analysis engine comprises: code that allows a user to read classfiles.

63. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one core library and at least one data analysis engine comprises: code that iteratively computes an effect of an instruction on information associated with locations on a method frame, said method frame being an operand stack and a local variables array.

64. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one core library and at least one data analysis engine comprises: code that computes the effect of a method on information associated with variables that still exist upon completion of this method.

65. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one utility module comprises: code that identifies a set of possible types for each instruction and each frame location in each method.

66. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one utility module comprises: code that identifies, for each method, a set of escaping objects and class variables from which a mutable object is reachable for each instruction and each frame location referring to said mutable object.

67. (Original) The computer system as recited in claim 60, wherein the code that

OCT 3 2005

maintains the layer of at least one mutability sub-analysis module comprises: code that identifies, for each method, a set of fields whose corresponding instance and class variables may be set within said each method.

68. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one mutability sub-analysis module comprises: code that identifies, for each method, a set of reference-type fields and method parameters, said set of reference-type fields and method parameters referencing an object, a state of said object being modified by said each method.

69. (Original) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one mutability sub-analysis module comprises: code that identifies, for each variable, whether its value may be modified directly by at least one method that is not within the program component.

70. (Previously Presented) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one mutability sub-analysis module comprises:

code that detects possible accessibility of a state of each object, by determining if each variable associated with said object is encapsulated,

wherein a variable is encapsulated if all references to objects reachable from it are defined within the program component and

said accessibility is made by at least one method that is not within the program component.

71. (Previously Presented) The computer system as recited in claim 60, wherein the code that maintains the layer of at least one mutability sub-analysis module comprises:

code that detects a possible breakage of encapsulation,

wherein a variable is encapsulated if all references to mutable objects reachable from it are defined within the program component and

variable encapsulation is broken if a method within the program component causes a

mutable object reachable from the variable to become accessible to at least one method that is not within the program component.

OCT 3 2005

REMARKS

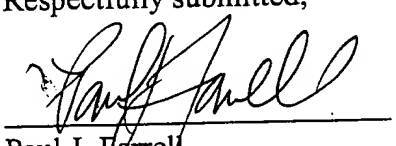
Claims 1-71 are pending in the application. All Claims have been rejected.

The Examiner rejected Claims 1-71 under 35 U.S.C. §101, as being directed to non-statutory subject matter. In response independent Claims 1, 7, 8, 19, 26, 38, 44, 45, 53, and 60 were amended to clarify that the program components recited in the claims are executing on a computing device which includes a processor and memory and that the claim steps are performed to identify an exposure of the variables of the program component to modification external to the program component. These amendments now make the subject matter statutory, as defined by State Street Bank & Trust Co.v. Signature Financial Group Inc., 149 F.3d 1368,47 USPQ2d 1596 (Fed.Cir.1998) and described in MPEP section 2106 titled "Patentable Subject Matter - Computer-Related Inventions".

Accordingly, it is submitted that independent Claims 1, 7, 8, 19, 26, 38, 44, 45, 53, and 60 are directed to statutory subject matter and, since the rejection in view of the prior art has been withdrawn by the Examiner it is believed that these claims are therefore patentable. Without conceding the patentability per se of dependent Claims 2-6, 9-25, 27-43, 46-52, 54-59, and 61-71, these are likewise believed to overcome rejection by virtue of their dependence on their respective independent claims.

Applicants believe that all Claims pending in the present application are in condition for allowance. If the Examiner has any questions regarding this communication or feels that an interview would be helpful in advancing the prosecution of this application, the Examiner is requested to contact the undersigned attorney.

Respectfully submitted,



Paul J. Farrell
Reg. No. 33,494
Attorney for Applicant

DILWORTH & BARRESE, LLP
333 Earle Ovington Boulevard
Uniondale, New York 11553
Telephone: 516-228-8484
Facsimile: 516-228-8516